

Contents

1		2
2	Kapitel 1: Einführung	3
	2.0.1 Übersicht über die Applikation	3
	2.0.2 Wie startet man die Applikation?	3
	2.0.3 Wie testet man die Applikation?	3
3	Kapitel 2: Clean Architecture	3
	3.0.1 Was ist Clean Architecture?	3
	3.0.2 Analyse der Dependency Rule	4
	3.0.3 Analyse der Schichten	4
4	Kapitel 3: SOLID	4
	4.0.1 Analyse Single-Responsibility-Principle (SRP)	4
	4.0.2 Analyse Open-Closed-Principle (OCP)	4
	4.0.3 Analyse Liskov-Substitution- (LSP), Interface-Segregation- (ISP), Dependency-Inversion-Principle (DIP)	5
5	Kapitel 4: Weitere Prinzipien	5
	5.0.1 Analyse GRASP: Geringe Kopplung	5
	5.0.2 Analyse GRASP: Hohe Kohäsion	5
	5.0.3 Don't Repeat Yourself (DRY)	5
6		6
7	Kapitel 5: Unit Tests	6
	7.0.1 10 Unit Tests	6
	7.0.2 ATRIP: Automatic	6
	7.0.3 ATRIP: Thorough	6
	7.0.4 ATRIP: Professional	6
	7.0.5 Code Coverage	6
	7.0.6 Fakes und Mocks	6
8	Kapitel 6: Domain Driven Design	7
	8.0.1 Ubiquitous Language	7
	8.0.2 Entities	7
	8.0.3 Value Objects	7
	8.0.4 Repositories	7
	8.0.5 Aggregates	7

9 Kapitel 7: Refactoring	7
9.0.1 Code Smells	7
9.0.2 2 Refactorings	9
10 Kapitel 8: Entwurfsmuster	9
10.0.1 Entwurfsmuster: [Name]	9
10.0.2 Entwurfsmuster: [Name]	9
Programmentwurf	
[Bezeichnung]	
Name: [Name, Vorname]	
Matrikelnummer: [MNR]	
Abgabedatum: [DATUM]	

1

Allgemeine Anmerkungen:

- es darf nicht auf andere Kapitel als Leistungsnachweis verwiesen werden (z.B. in der Form “XY wurde schon in Kapitel 2 behandelt, daher hier keine Ausführung”)
- alles muss in UTF-8 codiert sein (Text und Code)
- sollten mündliche Aussagen den schriftlichen Aufgaben widersprechen, gelten die schriftlichen Aufgaben (ggf. an Anpassung der schriftlichen Aufgaben erinnern!)
- alles muss ins Repository (Code, Ausarbeitung und alles was damit zusammenhängt)
- die Beispiele sollten wenn möglich vom aktuellen Stand genommen werden
 - finden sich dort keine entsprechenden Beispiele, dürfen auch ältere Commits unter Verweis auf den Commit verwendet werden
 - Ausnahme: beim Kapitel “Refactoring” darf von vorne herein aus allen Ständen frei gewählt werden (mit Verweis auf den entsprechenden Commit)
- falls verlangte Negativ-Beispiele nicht vorhanden sind, müssen entsprechend mehr Positiv-Beispiele gebracht werden

- Achtung: werden im Code entsprechende Negativ-Beispiele gefunden, gibt es keine Punkte für die zusätzlichen Positiv-Beispiele
- Beispiele
 - * “Nennen Sie jeweils eine Klasse, die das SRP einhält bzw. verletzt.”
 - Antwort: Es gibt keine Klasse, die SRP verletzt, daher hier 2 Klassen, die SRP einhalten: [Klasse 1], [Klasse 2]
 - Bewertung: falls im Code tatsächlich keine Klasse das SRP verletzt: volle Punktzahl ODER falls im Code mind. eine Klasse SRP verletzt: halbe Punktzahl
- verlangte Positiv-Beispiele müssen gebracht werden
- Code-Beispiel = Code in das Dokument kopieren

2 Kapitel 1: Einführung

2.0.1 Übersicht über die Applikation

[Was macht die Applikation? Wie funktioniert sie? Welches Problem löst sie/welchen Zweck hat sie?]

2.0.2 Wie startet man die Applikation?

[Wie startet man die Applikation? Welche Voraussetzungen werden benötigt? Schritt-für-Schritt-Anleitung]

2.0.3 Wie testet man die Applikation?

[Wie testet man die Applikation? Welche Voraussetzungen werden benötigt? Schritt-für-Schritt-Anleitung]

3 Kapitel 2: Clean Architecture

3.0.1 Was ist Clean Architecture?

[allgemeine Beschreibung der Clean Architecture in eigenen Worten]

3.0.2 Analyse der Dependency Rule

[(1 Klasse, die die Dependency Rule einhält und eine Klasse, die die Dependency Rule verletzt); jeweils UML der Klasse und Analyse der Abhängigkeiten in beide Richtungen (d.h., von wem hängt die Klasse ab und wer hängt von der Klasse ab) in Bezug auf die Dependency Rule]

1. Positiv-Beispiel: Dependency Rule
2. Negativ-Beispiel: Dependency Rule

3.0.3 Analyse der Schichten

[jeweils 1 Klasse zu 2 unterschiedlichen Schichten der Clean-Architecture; jeweils UML der Klasse (ggf. auch zusammenspielenden Klassen), Beschreibung der Aufgabe, Einordnung mit Begründung in die Clean-Architecture]

1. Schicht: [Name]
2. Schicht: [Name]

4 Kapitel 3: SOLID

4.0.1 Analyse Single-Responsibility-Principle (SRP)

[jeweils eine Klasse als positives und negatives Beispiel für SRP; jeweils UML der Klasse und Beschreibung der Aufgabe bzw. der Aufgaben und möglicher Lösungsweg des Negativ-Beispiels (inkl. UML)]

1. Positiv-Beispiel
2. Negativ-Beispiel

4.0.2 Analyse Open-Closed-Principle (OCP)

[jeweils eine Klasse als positives und negatives Beispiel für OCP; jeweils UML der Klasse und Analyse mit Begründung, warum das OCP erfüllt/nicht erfüllt wurde – falls erfüllt: warum hier sinnvoll/welches Problem gab es? Falls nicht erfüllt: wie könnte man es lösen (inkl. UML)?]

1. Positiv-Beispiel
2. Negativ-Beispiel

4.0.3 Analyse Liskov-Substitution- (LSP), Interface-Segregation- (ISP), Dependency-Inversion-Principle (DIP)

[jeweils eine Klasse als positives und negatives Beispiel für entweder LSP oder ISP oder DIP); jeweils UML der Klasse und Begründung, warum man hier das Prinzip erfüllt/nicht erfüllt wird]

[Anm.: es darf nur ein Prinzip ausgewählt werden; es darf NICHT z.B. ein positives Beispiel für LSP und ein negatives Beispiel für ISP genommen werden]

1. Positiv-Beispiel
2. Negativ-Beispiel

5 Kapitel 4: Weitere Prinzipien

5.0.1 Analyse GRASP: Geringe Kopplung

[jeweils eine bis jetzt noch nicht behandelte Klasse als positives und negatives Beispiel geringer Kopplung; jeweils UML Diagramm mit zusammenspielenden Klassen, Aufgabenbeschreibung und Begründung für die Umsetzung der geringen Kopplung bzw. Beschreibung, wie die Kopplung aufgelöst werden kann]

1. Positiv-Beispiel
2. Negativ-Beispiel

5.0.2 Analyse GRASP: Hohe Kohäsion

[eine Klasse als positives Beispiel hoher Kohäsion; UML Diagramm und Begründung, warum die Kohäsion hoch ist]

5.0.3 Don't Repeat Yourself (DRY)

[ein Commit angeben, bei dem duplizierter Code/duplizierte Logik aufgelöst wurde; Code-Beispiele (vorher/nachher); begründen und Auswirkung beschreiben]

6

7 Kapitel 5: Unit Tests

7.0.1 10 Unit Tests

[Nennung von 10 Unit-Tests und Beschreibung, was getestet wird]

Unit Test	Beschreibung
Klasse#Methode	

7.0.2 ATRIP: Automatic

[Begründung/Erläuterung, wie 'Automatic' realisiert wurde]

7.0.3 ATRIP: Thorough

[jeweils 1 positives und negatives Beispiel zu 'Thorough'; jeweils Code-Beispiel, Analyse und Begründung, was professionell/nicht professionell ist]

7.0.4 ATRIP: Professional

[jeweils 1 positives und negatives Beispiel zu 'Professional'; jeweils Code-Beispiel, Analyse und Begründung, was professionell/nicht professionell ist]

7.0.5 Code Coverage

[Code Coverage im Projekt analysieren und begründen]

7.0.6 Fakes und Mocks

[Analyse und Begründung des Einsatzes von 2 Fake/Mock-Objekten; zusätzlich jeweils UML Diagramm der Klasse]

8 Kapitel 6: Domain Driven Design

8.0.1 Ubiquitous Language

[4 Beispiele für die Ubiquitous Language; jeweils Bezeichnung, Bedeutung und kurze Begründung, warum es zur Ubiquitous Language gehört]

Bezeichnung	Bedeutung	Begründung
Link		
Category		

8.0.2 Entities

[UML, Beschreibung und Begründung des Einsatzes einer Entity; falls keine Entity vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

8.0.3 Value Objects

[UML, Beschreibung und Begründung des Einsatzes eines Value Objects; falls kein Value Object vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

8.0.4 Repositories

[UML, Beschreibung und Begründung des Einsatzes eines Repositories; falls kein Repository vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

8.0.5 Aggregates

[UML, Beschreibung und Begründung des Einsatzes eines Aggregates; falls kein Aggregate vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

9 Kapitel 7: Refactoring

9.0.1 Code Smells

[jeweils 1 Code-Beispiel zu 2 Code Smells aus der Vorlesung; jeweils Code-Beispiel und einen möglichen Lösungsweg bzw. den genommen Lösungsweg]

beschreiben (inkl. (Pseudo-)Code)]

1. Duplicated Code Da es bei Java keine Funktion zum durchsuchen eines Sets gibt wurde an mehreren Stellen ein Konstrukt, wie unten sichtbar verwendet. Dies macht den Code unleserlich und schwerer zu warten.

```
// LinkRepository.java
public Optional<Link> getById(LinkId id) {
    return links.stream().filter(link -> link.getId().equals(id)).findFirst();
}

public Optional<Link> getByUrl(LinkUrl url) {
    return links.stream().filter(link -> link.getUrl().equals(url)).findFirst();
}
```

Durch die Einführung des Dekorator-Entwurfsmuster für Set wurde jedoch auch eine eigene Implementation eines Sets eingeführt. Dadurch konnte diese Set Implementation auch einfach durch eine find Methode ergänzt werden, wie dargestellt.

```
// CustomStrictSet.java
@Override
public Optional<T> find(Predicate<T> predicate) {
    return set.stream().filter(predicate).findFirst();
}
```

So wurde die Codezeile an vier Stellen ersetzt. Wenn das Refactoring nicht recht früh durchgeführt worden wäre, wären es eventuell sogar mehr Stellen geworden.

```
// LinkRepository.java
public Optional<Link> getById(LinkId id) {
    return links.find(link -> link.getId().equals(id));
}

public Optional<Link> getByUrl(LinkUrl url) {
    return links.find(link -> link.getUrl().equals(url));
}
```

Das Refactoring wurde mit Commit e4f1670742 durchgeführt.

9.0.2 2 Refactorings

[2 unterschiedliche Refactorings aus der Vorlesung anwenden, begründen, sowie UML vorher/nachher liefern; jeweils auf die Commits verweisen]

10 Kapitel 8: Entwurfsmuster

[2 unterschiedliche Entwurfsmuster aus der Vorlesung (oder nach Absprache auch andere) jeweils sinnvoll einsetzen, begründen und UML-Diagramm]

10.0.1 Entwurfsmuster: [Name]

10.0.2 Entwurfsmuster: [Name]